正课:

1. 添加/删除:

2. HTML DOM 常用对象

3. BOM

window

打开关闭窗口

history

location

定时器

navigator

1. 添加删除:

优化: 尽量减少操作DOM树的次数

为什么: 减少重排重绘

如何: 2种:

1. 如果同时添加父元素和子元素，则应该现在内存中，将子元素添加到父元素，再最后将父元素一次性整体添加到DOM Tree

2. 如果父元素已经在页面上了，要添加多个平级子元素，则应该使用文档片段:

什么是文档片段: 内存中临时存储多个子元素的虚拟父元素

何时: 只要向网页中同时添加多个平级子元素时，都可用文档片段

如何: 3步:

1. 创建文档判断

2. 将子元素添加到文档片段

3. 将文档片段整体添加到DOM 树

删除: parent.removeChild(child)

通常: child.parentNode.removeChild(child)

2. HTML DOM常用对象:

Image: 创建: var img=new Image();

Select: 代表页面上一个select元素

属性:

.selectedIndex 获得当前select选中的option的下标位置

.value 获得select中选中的option的值

如果选中的option没有value属性，则用innerHTML代替

.options 获得当前select下所有option的集合

.options.length 获得select下所有option的个数

.options.length=0

.length => .options.length

清空所有option .length=0

方法: .add(option) 追加一个option

问题: .add不支持文档片段

.remove(i) 移除i位置的option

Option: 代表select下一个option元素

创建: var opt=new Option(text,value)

属性: .text .value .index

Table: 代表一个table元素

管着行分组:

创建行分组: var thead=table.createTHead()

var tbody=table.createTBody()  
 var tfoot=table.createTFoot()

删除行分组: table.deleteTHead()

table.deleteTFoot()

获取行分组: table.tHead table.tFoot table.tBodies[i]

行分组: 管着行:

添加行: var tr=行分组.insertRow(i)

在i位置插入一个新行

固定套路: 1. 在末尾追加新行: .insertRow()

2. 在开头插入新行: .insertRow(0)

删除行: 行分组.deleteRow(i) ——不常用!

问题: i要求是在当前行分组内的相对下标位置

无法自动获得

应换为: table.deleteRow(tr.rowIndex)

tr.rowIndex可自动获得当前行在整个table内的位置

获取行: 行分组.rows[i]

行: 管着格

添加格: var td=tr.insertCell(i)

固定用法: 行末尾追加一个格: tr.insertCell()

说明: 只能创建td，不能创建th

删除格: tr.deleteCell(i)

获取格: tr.cells[i]

Form: 代表一个form元素

获取: var form=document.forms[i/id]

属性: .elements 获得表单中所有表单元素的集合

.elements.length 获得表单中，表单元素的个数

.length => .elements.length

方法: .submit() 代替submit按钮，在程序中实现手动提交表单

表单元素:

获取: form.elements[i/id/name]

更简化: form.name

方法: .focus 让当前表单元素获得焦点

.blur

3. BOM: Browser Object Model

什么是: 专门操作浏览器窗口的API

没有标准, 导致浏览器兼容性问题

包括:

window: 2个角色:

1. 代替global充当全局作用域对象

2. 封装所有DOM和BOM的API

history:

location:

navigator:

document: 封装DOM树，并提供操作DOM的API

event

screen

window:

属性: .innerWidth, .innerHeight 浏览器窗口中，文档显示区的宽和高

方法: .open() .close()

.open("url","name")

三种: 1. 在当前窗口打开，可后退: .open("url","\_self")

2. 在新窗口打开，可打开多个: .open("url","\_blank")

3. 在新窗口打开，只能打开一个:

.open("url","自定义窗口名")

history: 保存当前窗口打开后，成功访问过的url的历史记录栈

在当前窗口中，每访问一个新url，都会将新url压入history

API: history.go(n)

3种: 前进: history.go(1) 后退: history.go(-1)

刷新: history.go(0)

location: 保存当前窗口正在打开的url的对象

属性:

.href 完整url地址

.protocol 协议

.host 主机名+端口号

.hostname 主机名

.port 端口号

.pathname 相对路径

.hash #锚点地址

.search ?查询字符串

方法:

1. 在当前窗口打开，可后退:

location.assign(url) => location.href=url => location=url

2. 在当前窗口打开，禁止后退:

location.replace(url)

3. 重新加载页面: 刷新: 2种:

1. 普通刷新:

优先从浏览器本地缓冲获取资源:

F5

history.go(0)

location.reload(/\*false\*/)

2. 强制刷新:

无论本地是否有缓存，总是强制从服务器获取资源

location.reload(true)

定时器: 2种:

1. 周期性定时器:

什么是: 让程序每隔指定的时间间隔，反复执行一项任务

何时: 只要让程序按照指定的时间间隔，自动执行一项任务

如何: 3件事:

1. 任务函数: 让定时器反复执行的任务

2. 启动定时器: timer=setInterval(task, interval)

让程序，每隔interval 毫秒自动执行一次task任务

3. 停止定时器: clearInterval(timer)

timer: 定时器的序号, 在内存中唯一标识定时器的整数

专门用于停止定时器

如何获得: 只能在启动定时器时获得。

何时: 只要一个定时器可能被停止，就要在启动时，先保存定时器序号

好的习惯: 在clearInterval之后，手动清除timer中残留的序号: timer=null;

停止定时器: 2种情况:

1. 用户手动停止:

2. 定时器可自动停止:

在任务函数中，设定临界值，如果没有达到临界值，则继续执行任务，否则，如果达到临界值，就自动调用clearInterval

2. 一次性定时器:

什么是: 让程序先等待一段时间，再执行一次任务。执行后，自动停止。

何时: 只要让程序延迟执行一件事时

如何: 3件事:

1. task

2. 启动定时器: timer=setTimeout(task,wait)

让程序等待wait毫秒后，自动执行一次task，执行后自动停止

3. 停止定时器: clearTimeout(timer)